Autonomous Robots
https://doi.org/10.1007/s10514-019-09829-4

Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control

Colin Greatwood[1,2] · **Arthur G. Richards**[1,2]

Received: 23 May 2017 / Accepted: 2 January 2019

Abstract
A new method for enabling a quadrotor micro air vehicle (MAV) to navigate unknown environments using reinforcement learning (RL) and model predictive control (MPC) is developed. An efficient implementation of MPC provides vehicle control and obstacle avoidance. RL is used to guide the MAV through complex environments where dead-end corridors may be encountered and backtracking is necessary. All of the presented algorithms were deployed on embedded hardware using automatic code generation from Simulink. Results are given for flight tests, demonstrating that the algorithms perform well with modest computing requirements and robust navigation.

Keywords Model predictive control · Reinforcement learning · Exploration · Micro air vehicle

1 Introduction

This paper introduces a method for navigation and control of quadrotors within a non-convex obstacle field. The method uses online optimization within a model predictive control (MPC) framework, taking advantage of Fast MPC (Wang and Boyd 2010) with soft constraint modifications (Richards 2015) to provide a real-time controller on embedded hardware. Furthermore, the use of reinforcement learning (RL) enables autonomous navigation by providing high level path planning decisions for navigation of previously unexplored spaces. Flight test experiments demonstrate the methods within a two dimensional control scenario. The experiments use off-board localization by motion capture and synthesized sensing of obstacles: although these also include important challenges, the focus here is on the decision-making.

Trajectory generation in the presence of obstacles is NP-hard (Reif 1979) and has been the subject of considerable algorithm development, including randomized methods (LaValle and Kuffner 1999; Garcia and How 2005), integer programming (Richards and How 2002) and nonlinear optimization (Milam et al. 2000; Borrelli et al. 2006; Cowling et al. 2010). Another approach has been to separate the problem into multiple planning layers, for example combining the travelling salesman problem and potential field methods (Nieuwenhuisen et al. 2014).

The MPC framework introduced in this paper adopts a two stage process to avoid high computational requirements. Like Augugliaro et al. (2012), Deits and Tedrake (2015) and Sharma (2011), a local, convex optimization problem is derived from the harder global problem. Other approaches along these lines include following tunnels (Vitus et al. 2008), receding horizon optimization (Bellingham et al. 2002) or combining path generation with dynamic optimization (Hoffmann et al. 2008) or feasibility testing (Hehn and D'Andrea 2011). The authors' approach here is to decompose the problem geometrically to form a local convex problem and then deploy a quadratic program (QP) optimization. This is similar in spirit to the ellipsoidal tunneling method by Sharma (2011) but without requiring quadratic constraints.

MPC provides a framework to unify control, including stability and robustness analysis, with motion planning, including dynamics and operating constraints (Maciejowski 2002). Liu and Chen (2013) illustrated its potential for UAV control with obstacle avoidance constraints. Considerable work has focussed on tailoring fast real-time optimizers for MPC, of which the work by Wang and Boyd (2010) is adopted

✉ Arthur G. Richards
arthur.richards@bristol.ac.uk

Colin Greatwood
colin.greatwood@bristol.ac.uk

[1] Department of Aerospace Engineering, University of Bristol, Queens Building, University Walk, Bristol BS8 1TR, United Kingdom

[2] Bristol Robotics Laboratory, Bristol, United Kingdom

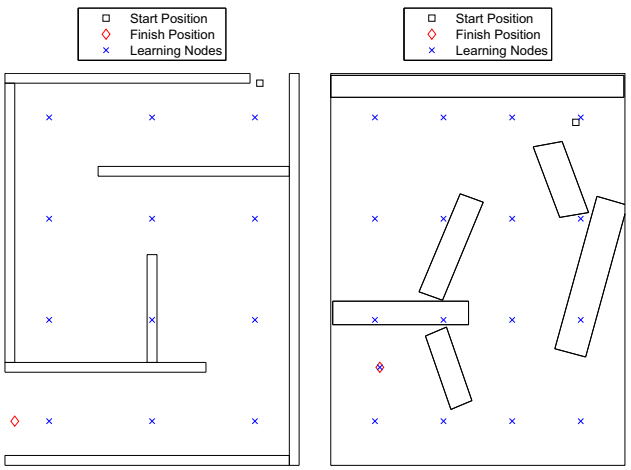
Published online: 12 January 2019

here. Faster solvers can be implemented using FPGAs (Hartley et al. 2014) but this is beyond the scope of this paper.

Quadrotors are popular test beds for autonomous vehicle research. Instrumented flying facilities have been developed, including MIT's Raven (How et al. 2008), Stanford's STARMAC (Hoffman et al. 2004), Pennsylvania's GRASP lab (Michael et al. 2010) and ETH's Flying Machine Arena (Hehn and D'Andrea 2011), all actively researching autonomous capabilities for quadrotors and other vehicles. Many suitable airframes are available and existing research addresses how to stabilize them and follow a path to a destination. For this work, a key question is "where to go next?" if no prior map of the world is provided and only local sensing is available. Environments such as those depicted in Fig. 1 pose a difficult problem to the MAV, where dead ends in the environment would make it easy for a local planner to become stuck.

Planning within environments with uncertain maps also poses challenges. For example, when using simultaneous localisation and mapping (SLAM) the robot's map of the world can become distorted and relies on methods such as loop closure (Williams et al. 2009) to re-align features in the map. A key feature of the proposed method is that it does not depend on an accurate global map for far-term planning, unlike other MPC-based methods (Bellingham et al. 2002). In principle, this makes the method robustly compatible with different mapping and localization strategies.

Autonomous exploration has been demonstrated in the past using frontier-based mapping (Yamauchi 1997), where an evidence grid is formed in order to map locations that are occupied by obstacles. This relies on maintaining a global map of the obstacle locations, but has shown good performance in complex and cluttered environments. Other methods such as the subsumption architecture (Brooks 1986) for boundary tracing and mapping (Mataric 1992) enable the robot to map an area whilst avoiding people moving within its environment. For more variety of sparsity in the environment Fraundorfer et al. (2012) use a combination of frontier-based mapping and the bug algorithm (Choset et al. 2005). The common theme between these methods is that they perform exploration by maintaining a map of the locations of the environment's obstacles.



(a) Obstacle field 1 **(b)** Obstacle field 2

Fig. 1 Example scenarios

Exploration of previously unmapped environments is addressed here by developing a reinforcement learning (Sutton and Barto 1998) (RL) algorithm. RL is a machine learning technique that updates its knowledge about the world based upon rewards following actions taken. A discrete set of node locations in the world are given a weighting, or *cost*, and the RL algorithm targets the lowest cost node visible. The obstacle avoidance is handled separately by the MPC controller. Should the MAV be unable to make progress, for example by spending time stuck in a dead end, it will learn to turn back and explore a different path. The full explanation of the control architecture is given in Sect. 3.

Other works have explored different combinations of MPC, machine learning, and UAVs. Zhang et al. (2016) used MPC as the supervisor for their learning algorithm, resulting in a deep neural network policy for obstacle avoidance, while Aswani et al. (2013) used a learning 'oracle' to refine the prediction model for a general MPC. Sharma and Taylor (2012) used RL for waypoint generation to improve the performance of their avoidance algorithm based on local ellipsoid constraints (Sharma 2011). Learning for exploration has been used in previous work with random neural Q-learning (Yang et al. 2016) to navigate and avoid obstacles in unknown environments. This approach to use learning for exploration has similarities to the work here, one of which is that it provides continuous control whilst discrete action choices about where to go next must be made. In (Yang et al. 2016) a continuous action space is built from the learning.

The distinctions of the present paper are that MPC is used online, not purely as part of the training. RL is used solely for waypoint selection, hence for a discrete choice rather than continuous, and compensates for the susceptibility of the locally convex MPC to explore local minima. The benefits of using RL for exploration to perform autonomous exploration is that an accurate global map does not need to be constructed, stored and maintained. An evidence grid, for example, could grow quickly if exploring large areas, especially if over three dimensions. It would also be necessary to produce a grid of fine enough a resolution such that the MAV could pass through narrow openings. Perhaps an even greater concern is one of robustness; it is well known (Lu and Milios 1997) that maintaining good alignment of a map is challenging and so any exploration technique that relies on a well maintained global map could fail.

Fig. 2 AR.Drone quadrotor

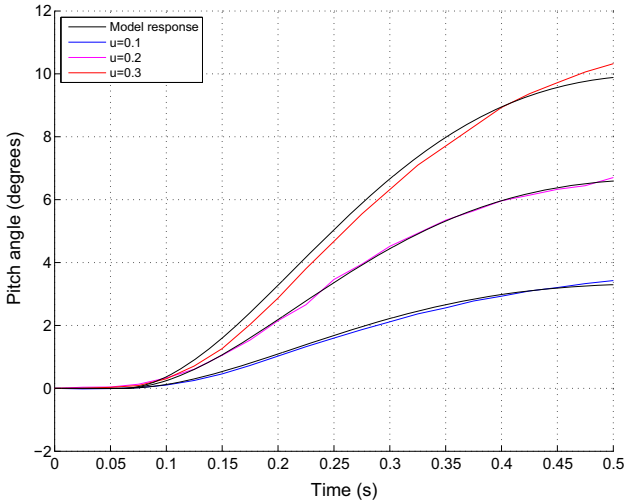


Fig. 3 Comparison of AR.Drone pitch flight data with model responses

2 Experimental setup and modelling

Experiments were performed using the Parrot AR.Drone (Fig. 2) within the Bristol Robotics Laboratory's flying arena. The flying arena is instrumented with ten Vicon motion capture system cameras, providing vehicle position information at 100Hz. The example obstacle fields (Fig. 1) were laid out over a six by eight metre area within the arena. The obstacle locations were defined numerically and detection is simulated through the knowledge of the vehicle's position within the Vicon coordinate system. All of the algorithms presented were programmed in MATLAB Simulink, compiled using automated code generation and executed on a dSpace MicroAutoBox. The MicroAutoBox has a 900Mhz PowerPC processor and enables rapid prototyping of software for execution on embedded hardware.

All of the development and results presented use an AR.Drone 2.0, which is shown in Fig. 2. This MAV is a popular off-the-shelf hobbyist platform, designed to be controlled via smart phones. Despite the low price and target market, the platform is well engineered and performs well for high level control algorithm development. The AR.Drone SDK was used to develop a bespoke controller application that converts UDP network commands from the MicroAutoBox into control signals for the AR.Drone, sent via WiFi.

In the scenario of exploration, speed is limited by the rate of decision-making, and the full agility of the quadrotor is not exploited and it remains close to hover configuration throughout. Therefore, we adopt the approach of Fraundorfer et al. (2012) and use a linear dynamics model determined by system identification techniques. The work uses level flight without rotation in yaw, controlled by downward-looking ultrasound, rate gyro and magnetometer, so only the lateral movement dynamics are considered. Feedback linearization (Helwa and Schoellig 2016) provides an alternative route to a linear model that would exploit more of the flight envelope, but this has not been pursued in the scope of this work.

The quadrotor was flown automatically in a hover under closed loop position control using Vicon for feedback. After settling in its starting position the AR.Drone's "trimmed" control inputs were frozen at their last given input but with a small step input superimposed on the pitch control axis. This caused the AR.Drone to pitch and move forward due to a known step input. A second order transfer function with delay was subsequently fitted to the measured pitch angle response using a Nelder–Mead optimisation (Nelder and Mead 1965). The pitch angle was measured using the Vicon system, which tracks all six degrees of freedom of the drone. The optimisation minimised the square of the difference between the first half a second of the pitch response in radians and the transfer function $G_\theta(s)$, finding

$$G_\theta(s) = e^{-0.06s} \frac{0.51}{0.017s^2 + 0.14s + 1} \tag{1}$$

The fit of the transfer function $G_\theta(s)$ to the measured pitch response is shown in Fig. 3, demonstrating a good match for the different control input values. It was found that the input was saturated by the AR.Drone at 0.37. There are no units for this quantity: this command is a digital signal sent from the MicroAutoBox to the AR.Drone controller application.

The acceleration of the quadrotor may be approximated as being acceleration due to gravity multiplied by the pitch angle, which is appropriate for small angles and low speeds before drag becomes a dominant force. The transfer functions describing the position and velocity dynamics may therefore be written as

$$G_{pos}(s) = \frac{g}{s^2} G_\theta(s) \tag{2}$$

$$G_{vel}(s) = \frac{g}{s} G_\theta(s) \tag{3}$$

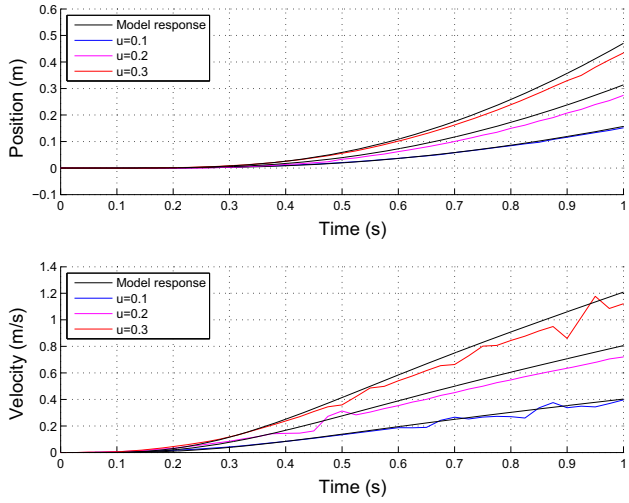


Fig. 4 Comparison of position and velocity flight data with model responses

respectively. A comparison of the above transfer functions and the measured responses is shown in Fig. 4 for different control inputs.

As will be introduced in Sect. 5, the MPC formulation requires a discretized linear state space representation of the vehicle, where the discrete state progression is written as

$$x(k+1) = Ax(k) + B\Delta u(k) + w(k) \tag{4}$$

$$y(k) = Cx(k) \tag{5}$$

where $x(k)$ is the state at time step k, $u(k)$ is the control input, $w(k)$ is an unknown disturbance and $y(k)$ is the output. From Eq. 1, for a single axis (pitch or roll) operating at 10 Hz the system matrices A_1 and B_1 are

$$A_1 = \begin{bmatrix} 1.0000 & 0.1000 & 0.0048 & 0.0001 & 0.0010 \\ 0 & 1.0000 & 0.0921 & 0.0037 & 0.0397 \\ 0 & 0 & 0.7815 & 0.0614 & 1.0980 \\ 0 & 0 & -3.6435 & 0.2692 & 18.3076 \\ 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{6}$$

$$B_1 = [0, 0.0029, 0.2095, 9.9251, 1.0000]^T \tag{7}$$

and for the complete model of the drone moving in 2D

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_1 \end{bmatrix} \tag{8}$$

$$B = \begin{bmatrix} B_1 & 0 \\ 0 & B_1 \end{bmatrix} \tag{9}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{10}$$

with the state vector is defined as

$$x = \left(p_x \ v_x \ \theta \ \dot{\theta} \ u_x \ p_y \ v_y \ \phi \ \dot{\phi} \ u_y \right)^T . \tag{11}$$

The output is the position $y = (p_x, p_y)$ while the other states include the velocity (v_x, v_y), pitch θ, roll ϕ and their derivatives, and the stored controls $(u_x(k), u_y(k))$.

3 Exploration architecture

The outline of the MAV control scheme used in this work is shown in Fig. 5. Directly controlling the movement of the MAV is an MPC controller, which will be introduced in Sect. 5. The MPC controller commands the MAV to move to setpoints provided. The state $x(k)$ of the vehicle is estimated by the state estimation block, using a Luenberger observer driven by the position measurements (5). To avoid the complexity of a Kalman filter with an augmented state, disturbance estimation is performed by low-pass filtering the measured prediction error (Tatjewski 2014):

$$\tilde{w}(k) = \hat{x}(k) - A\hat{x}(k-1) - B\Delta u(k-1) \tag{12}$$

where $\hat{x}$ is the estimated state. Both of these estimations are used by the MPC block in order to compute optimal control inputs. The "Convexification" block determines a convex obstacle-free operating region forming the constraints for the MPC. A target position $y_S(k) = \left(p_x^S(k), p_y^S(k)\right)^T$ is also provided to the MPC controller. The calculation of the convex operating region and provision of the setpoint are described in Sect. 4. This paper assumes fixed altitude and hence 2-D motion.

The reinforcement learning block uses temporal difference learning to determine a favourable local target or "node" to aim for, rather than simply aiming for a final global goal location. By doing so, the controller may guide the MAV through a non-convex space without getting stuck in dead ends. The learning algorithm block is described in Sect. 6.

The MAV position is measured externally via a Vicon motion capture system, which is passed into a virtual sensor block. The virtual sensor block uses the measured position and returns obstacle face information based upon hard-coded obstacle fields such as those described in Fig. 1. In future work, this block could be swapped for a real sensor so that obstacles would not have to be pre-defined in code. Also, goal locations would be based on recognition, not location. However, here we focus on control aspects, so simplifications are adopted in sensing.

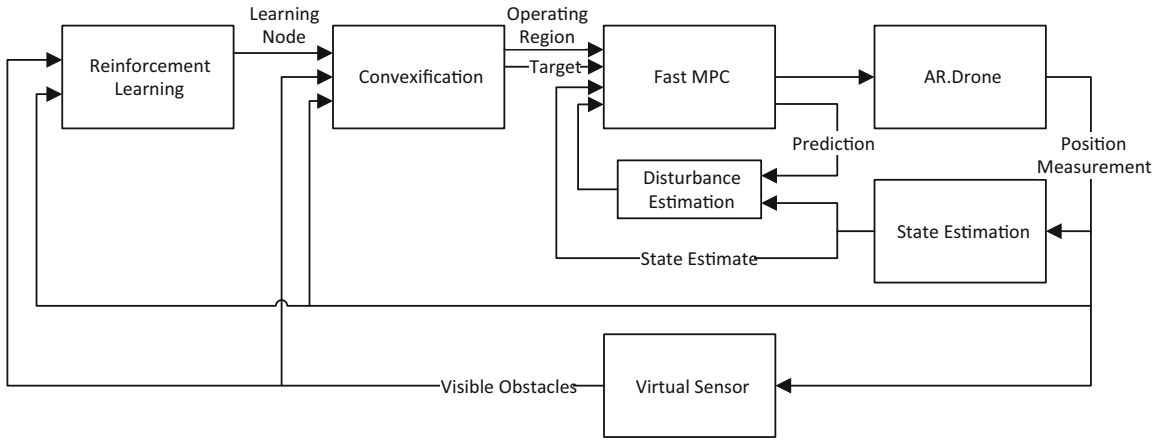


Fig. 5 Controller overview

4 Operating region calculation

The MPC method used employs quadratic programming (QP), which requires a convex solution space to be described in terms of linear constraints. The non-convex obstacle field must therefore also be decomposed into a convex solution space, or rather a convex 'operating region' within which the MAV is allowed to fly.

Vitus et al. (2008) propose two methods for decomposing a non-convex solution space into a sequence of convex polytopes in their paper on tunnel mixed integer linear programming (MILP). These are trapezoidal decomposition and Delaunay triangulation. This reduces the complexity of the online optimization but it still requires MILP. Augugliaro et al. (2012) went further and reduced the local problem to a local quadratic optimization by linearizing multi-vehicle separation constraints. Similarly, Sharma (2011) reformulated the region determination as a QP by adopting ellipsoidal regions. Deits and Tedrake (2015) present an iterative semidefinite programming method for finding convex regions that might also be suitable for defining the operating region within which the quadrotor may fly.

Two methods for computing convex operating regions from the non-convex obstacle fields are presented here. The first method provides rectangular operating regions with limits aligned with the global x and y axes, which for the purposes of the presented experiments will also always align with the vehicle x and y axes. The second method provides operating regions that also have four faces, but the faces are orientated in an effort to increase freedom in the direction of travel. Both methods provide operating regions that impose a fixed number of position constraints upon the MAV's movement, which is desirable to avoid having to rebuild the MPC.

4.1 Rectangular operating regions

The rectangular convexification method presented here computes operating regions with orthogonal faces. The method therefore assumes all obstacles are rectangular or expanded to become rectangular. By aligning the obstacle and hence operating region faces with the x and y axes, it is possible to pose the MPC as two separate MPCs with one for each axis. Decoupling the MPCs is possible due to the symmetry of the MAV. It is anticipated that two small MPCs will be less computationally expensive than a single large one. The computational savings are identified in Sect. 7.

First, define the operating region $\mathcal{R}$ as a rectangle

$$\mathcal{R} = \{(p_x, p_y) \in [x^{\min}, x^{\max}] \times [y^{\min}, y^{\max}]\} \tag{13}$$

where the position limits $(x^{\min}, x^{\max}, y^{\min}, y^{\max})$ are found using Algorithm 1.

The algorithm for finding the operating region starts by defining the distance d that denotes the distance the MAV could travel if it were travelling at the maximum permissible velocity $(v_{\lim})$ for all T time steps. The operating region limits are then defined by the MAV position p plus or minus the distance d. The newly defined operating region $\mathcal{R}$ is then inspected to see if any of the obstacle faces $\mathcal{F}$ intersect, *i.e.* the operating region is not free of obstacles. Should any of the obstacle faces intersect the operating region, the operating region is shrunk until it is free of obstacles.

4.2 Quadrilateral operating regions

The rectangular convexification method presented in Sect. 4.1 assumed the x and y axes are decoupled, which is reasonable for the dynamics of the MAVs used. This assumption will, however, restrict the complexity of operating region shapes

Algorithm 1 Rectangular Convexification

$d \leftarrow v_{\text{lim}} \cdot T\Delta t$
$x^{\min} \leftarrow p_x - d$
$x^{\max} \leftarrow p_x + d$
$y^{\min} \leftarrow p_y - d$
$y^{\max} \leftarrow p_y + d$
while $\mathcal{R} \cap \mathcal{F} \neq \emptyset$ **do**
  if Obstacle corner $[C_x, C_y]$ is inside $\mathcal{R}$ **then**
    if $|p_x - C_x| < |p_y - C_y|$ **then**
      Shrink $\mathcal{R}$ in y
    else
      Shrink $\mathcal{R}$ in x
    end if
  end if
  if Obstacle face is inside $\mathcal{R}$ **then**
    if Obstacle face is aligned with y **then**
      Shrink $\mathcal{R}$ in x
    else if Obstacle face is aligned with x **then**
      Shrink $\mathcal{R}$ in y
    else
      Shrink $\mathcal{R}$ in x and y
    end if
  end if
end while

that can be used. The second method as described here lifts this requirement, which could provide more freedom for the MAV's movement when operating close to obstacles that are not aligned with the vehicle's axes.

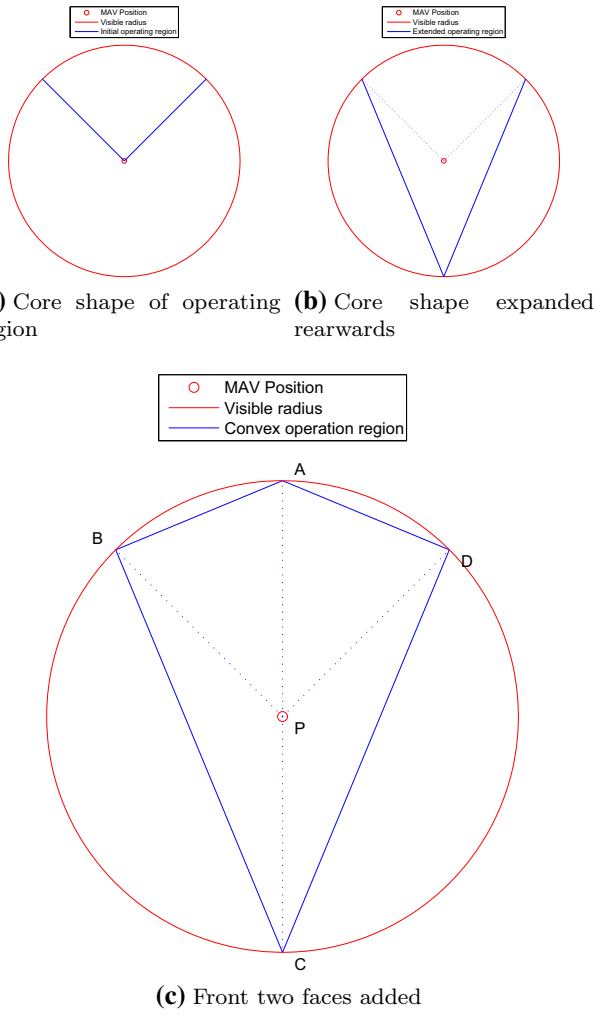
The virtual sensor block allows the MAV to see obstacles at a distance of up to 2 metres. The process of defining the operating region consists of many stages, whereby an initial region that fits within the 2 metre radius is reduced in size until it no long intersects detected obstacles.

In order to maximize manoeuvrability in the direction of travel, the basic shape of the operating region would be a cone, expanding from the current position. This is shown in Fig. 6a.

To allow some margin for error in the position of the MAV, the region is then expanded rearwards, as shown in Fig. 6b. Finally, two faces are added to the front of the operating region, as shown in Fig. 6c. This forms the basic shape of the operating region; this is subsequently reduced in size until no obstacles are enclosed within the region. Algorithm 2 describes how the operating region is reduced in size in the presence of obstacle faces. Each step in the algorithm produces a subset (or identical copy) of the previous operating region and so never undoes earlier work.

5 Application of model predictive control

Model predictive control (Maciejowski 2002) is used to drive the quadrotor to setpoints determined by the planner while respecting operating constraints on velocity and control. The operating constraints also include the position constraints



(a) Core shape of operating region **(b)** Core shape expanded rearwards

(c) Front two faces added

Fig. 6 Basic shape of operating region

imposed by the convexification scheme presented in the previous section. The algorithm implemented uses the "Fast MPC" formulation of Wang and Boyd (2010) to achieve fast solution times.

In theory, it is possible to treat this scenario as a regulation problem, taking the setpoint position as the origin and converting all states to a setpoint-relative frame before passing to the MPC. However, this was found to give significant problems when the setpoint changed, often leading to loss of feasibility of the interior point optimizer. Instead, the method of Limon et al. (2008) has been adopted, treating the problem as one of tracking a piecewise-constant setpoint, using the modifications of Maeder et al. (2009) to give offset-free tracking in the presence of disturbance. The linear state space model from Sect. 2 is employed. The input change is subject to a hard constraint which must always be respected

$$F_x x(k) + F_u \Delta u(k) \leq f \tag{14}$$

Algorithm 2 Quadrilateral Convexification

Construct the basic operation region, as shown in Fig. 6(c)
if $\overrightarrow{AP}$ intersects a face **then**
  Move A along $\overrightarrow{AP}$ until there is no intersection
  if $\overrightarrow{AB}$ intersects a face **then**
    Move B along $\overrightarrow{BP}$ until there is no intersection
  end if
  if $\overrightarrow{AD}$ intersects a face **then**
    Move D along $\overrightarrow{DP}$ until there is no intersection
  end if
end if
if $\overrightarrow{BP}$ intersects a face **then**
  Move B along $\overrightarrow{BP}$ until there is no intersection
end if
if $\overrightarrow{CP}$ intersects a face **then**
  Move C along $\overrightarrow{CP}$ until there is no intersection
end if
if $\overrightarrow{DP}$ intersects a face **then**
  Move D along $\overrightarrow{DP}$ until there is no intersection
end if
if $\overrightarrow{AB}$ intersects a face **then**
  Move B to point of intersection
end if
if $\overrightarrow{AD}$ intersects a face **then**
  Move D to point of intersection
end if
if $\overrightarrow{BC}$ intersects a face **then**
  Move C along $\overrightarrow{CP}$ until there is no intersection
end if
if $\overrightarrow{CD}$ intersects a face **then**
  Move C along $\overrightarrow{CP}$ until there is no intersection
end if

$$\text{Where, } F_x = \begin{bmatrix} 0 & 0 \\ 0 \cdots & 0 \\ 0 \cdots & 0 \\ 0 & 0 \end{bmatrix} \tag{15}$$

$$F_u = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \tag{16}$$

$$f = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix} \tag{17}$$

Soft constraints (Kerrigan and Maciejowski 2000) are also incorporated, which must be respected if possible but can be violated if no alternative exists

$$F_{Sx}x(k) + F_{Su}\Delta u(k) \leq f_S \tag{18}$$

These soft constraints limit the quadrotor's position (to stay in the operating region for obstacle avoidance), maximum speed and absolute control input. The soft constraint matrices are therefore

$$F_{Sx} = \begin{bmatrix} 100 & 0 & 0\,0 & 0 & 0 & 0 & 0\,0 & 0 \\ -100 & 0 & 0\,0 & 0 & 0 & 0 & 0\,0 & 0 \\ 0 & 100 & 0\,0 & 0 & 0 & 0 & 0\,0 & 0 \\ 0 & -100 & 0\,0 & 0 & 0 & 0 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 100 & 0 & 0 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & -100 & 0 & 0 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 0 & 100 & 0 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 0 & -100 & 0 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 0 & 0 & 100 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 0 & 0 & -100 & 0\,0 & 0 \\ 0 & 0 & 0\,0 & 0 & 0 & 0 & 0\,0 & 100 \\ 0 & 0 & 0\,0 & 0 & 0 & 0 & 0\,0 & -100 \end{bmatrix} \tag{19}$$

$$F_{Su} = \begin{bmatrix} 0 \end{bmatrix} \tag{20}$$

$$f_S = \begin{bmatrix} 500 \\ 500 \\ 50 \\ 50 \\ 35 \\ 35 \\ 500 \\ 500 \\ 50 \\ 50 \\ 35 \\ 35 \end{bmatrix} \tag{21}$$

The MPC optimizer solves a quadratic program (QP), at each time step, to find the input sequence $U(k) = (\Delta u(k), \ldots, \Delta u(k+T))$ that minimizes a cost defined as

$$J = \sum_{j=0}^{T} (x(k+j) - x_S(k))^T Q (x(k+j) - x_S(k)) + \Delta u(k+j)^T R \Delta u(k+j) \tag{22}$$

where the state and input cost matrices Q and R are defined as

$$Q_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0 & 0.02 \end{bmatrix} \tag{23}$$

$$R_1 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{24}$$

$$Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_1 \end{bmatrix} \tag{25}$$

$$R = \begin{bmatrix} R_1 & 0 \\ 0 & R_1 \end{bmatrix} \tag{26}$$

The current target state $x_S(k)$ is found by solving the following equations (Maeder et al. 2009) for a disturbance-invariant state at the target position $y_S(k)$:

$$\begin{bmatrix} I - A \\ C \end{bmatrix} x_S(k) = \begin{pmatrix} \hat{w}(k) \\ y_S(k) \end{pmatrix} \tag{27}$$

where $\hat{w}(k)$ is a disturbance estimate found from the prediction error. The tracking cost (22) can be rewritten in the equivalent form

$$J = \sum_{j=0}^{T} x(k+j)^T Q x(k+j) + \Delta u(k+j)^T R \Delta u(k+j) - 2x_S(k)^T Q x(k+j) \tag{28}$$

where the constant term has been omitted, since it makes no difference to the optimization. Setting the final linear cost weight $-2x_S(k)^T Q = q^T$ puts the problem in the same form as for Fast MPC (Wang and Boyd 2010) and the reader is directed to that paper for details on how the QP is solved, including fast computation of the Newton step. The soft constraints are implemented using a penalty function and the complete solver is described in Richards (2015).

One significant modification is made to the MPC formulation from Wang and Boyd (2010) in order to enforce terminal equality constraints. For the quadrotor, it is convenient and effective to require the terminal velocity to be zero, in accordance with the idea of "safe" receding horizon control (Schouwenaars et al. 2004).

$$\begin{bmatrix} 0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \end{bmatrix} x(k+T+1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{29}$$

Fortunately, the addition of terminal equality constraints does not change the sparsity patterns exploited for fast computation by Wang and Boyd (2010). Note that it is only required that the terminal state of the plan be stationary, and since the plan is re-optimized before it is fully executed, the quadrotor rarely stops in practice until it reaches its current target $y_S(k)$. This will be seen in the results. The details of the modification are given in Appendix A. The MPC optimizer was programmed in a Simulink "embedded MATLAB function" for use with automatic code generation.

Since the hard constraints (14) only apply to the inputs, feasibility is guaranteed, without necessarily satisfying the state constraints. However, since the method inherits the properties from Maeder et al. (2009) and Limon et al. (2008), stability and satisfaction of *all* constraints is guaranteed provided that the soft constraints are sufficiently weighted (Kerrigan and Maciejowski 2000), the disturbance is constant and the constraints do not change. The dynamic convexification invalidates this last condition, although more recent methods (Bali and Richards 2017) provide a constrained convexification to promote recursive feasibility.

6 Reinforcement learning for navigation

Reinforcement learning (RL) is a machine learning technique that is employed here to help the exploration algorithms become 'unstuck' from dead ends and even unforeseen problems such as failures of the QP to converge. RL updates its knowledge about the world based upon rewards following actions taken. The MAV may therefore learn from time spent trying to progress via a dead end that it needs to turn back and explore a different path. The previous sections described the MPC and operating region calculation methods required to guide the MAV to a target position. By itself, however, the MPC is unable to guide the MAV throughout environments such as the example scenarios without leaving the MAV stuck down a dead end. To fix this problem, the RL algorithm dynamically chooses the target position $y_S(k)$. The convexification and MPC algorithms then provide the control inputs for transition to the latest target.

Reinforcement learning has been used in the past (Richards and Boyle 2010) to learn the cost-to-go for a receding horizon planner over successive repetitive UAV missions. The work in this paper, however, aims to use RL to enable the MAV to robustly navigate previously unexplored environments. Specifically, temporal difference learning (Sutton and Barto 1998) is employed.

The RL algorithm initializes by defining a grid of nodes $\mathcal{N}$ where each node $i \in \mathcal{N}$ is associated with a position (p_x^i, p_y^i). Each node is then given a dynamic cost $J^i(k)$, representing the time to reach the goal from its position, as estimated at time step k. The initial cost of each node is estimated using the distance from the final goal position (p_x^G, p_y^G) divided by the expected mean speed $\bar{v}$:

$$J^i(0) = \frac{\sqrt{(p_x^G - p_x^i)^2 + (p_y^G - p_y^i)^2}}{\bar{v}}. \tag{30}$$

During flight, the RL algorithm determines the target node $N(k) \in \mathcal{N}$ for the MAV to fly towards. The setpoint is selected greedily by considering all nodes that the MAV is currently able to see. A cost-to-go V^i is associated with flying to node i. This is computed from the cost held by the node and the estimated time required to reach the node.

$$V^i(k) = J^i(k) + d^i(k) \tag{31}$$

$$d^i(k) = \frac{\sqrt{(p_x(k) - p_x^i)^2 + (p_y(k) - p_y^i)^2}}{\bar{v}} \tag{32}$$

where $d^i(k)$ represents the straight line estimated flight time to node i. The node with the lowest cost-to-go is selected as $N(k)$ and passed to the MPC algorithm to guide the MAV.

Throughout the flight, the RL algorithm also needs to update the cost estimated for each node. This is the necessary step to learning information about the environment that the MAV is exploring. Every time the MAV is instructed to fly towards a new node, the time spent attempting to reach it is recorded. This time taken is used to update the cost for the target node in what is called temporal difference learning. The node costs are updated at discrete time steps. The cost of the node from the previous time step ($J^{N(k-1)}$) is updated as follows

$$J^{N(k-1)}(k) = J^{N(k-1)}(k-1) + \alpha\left(\Delta t + \frac{d^{N(k)}(k) - d^{N(k-1)}(k-1)}{\bar{v}} + J^{N(k)}(k-1) - J^{N(k-1)}(k-1)\right) \quad (33)$$

where each time step is Δt seconds long. The constant α is a weighting that is used to tune the rate of learning; a value of 0.8 was found to work well. It is possible for nodes $N(k)$ and $N(k-1)$ to point at the same node, which will indeed be the case for most time steps, and hence the updated cost of the node is based upon time spent aiming for it. It follows from (33) that the cost will be unchanged if the MAV gets closer to the same node at the estimated speed $\bar{v}$. However, if the MAV becomes stuck at a node, that nodes cost will increase. Subsequently, when a node is reached at a dead end location, the MAV will loiter for a few seconds whilst the cost builds up and it becomes 'cheaper' for it to turn around and backtrack.

7 Results

Section 7.1 provides simulation results for the Fast MPC algorithm, demonstrating the performance in both constraint satisfaction and computation time. Details of particular parts of the optimiser construction are discussed along with their impact on the results. Section 7.2 presents results for flight tests performed using all of the developed algorithms.

7.1 MPC performance

First, simulation results are presented in order to describe key features of the optimiser's construction. Section 7.1.1 presents the best case results for a one dimensional moving setpoint demonstration using a numerical model of the AR.Drone. These results are then used for comparison in Sect. 7.1.2 where the computation cost of two separate MPC optimisations is compared to one larger MPC optimisation. Section 7.1.3 then presents flight test results using the MPC for control of the AR.Drone with the learning disabled.

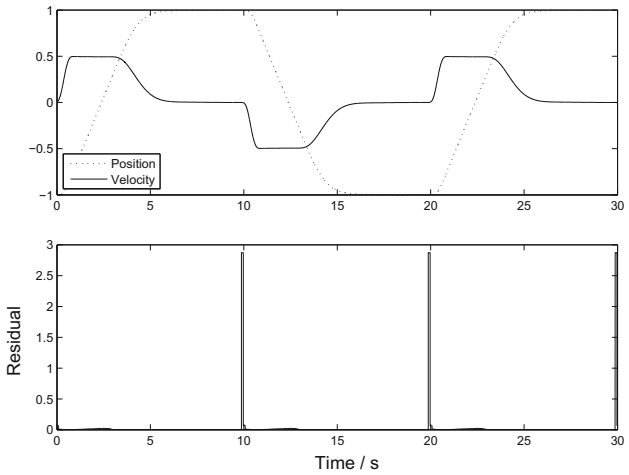


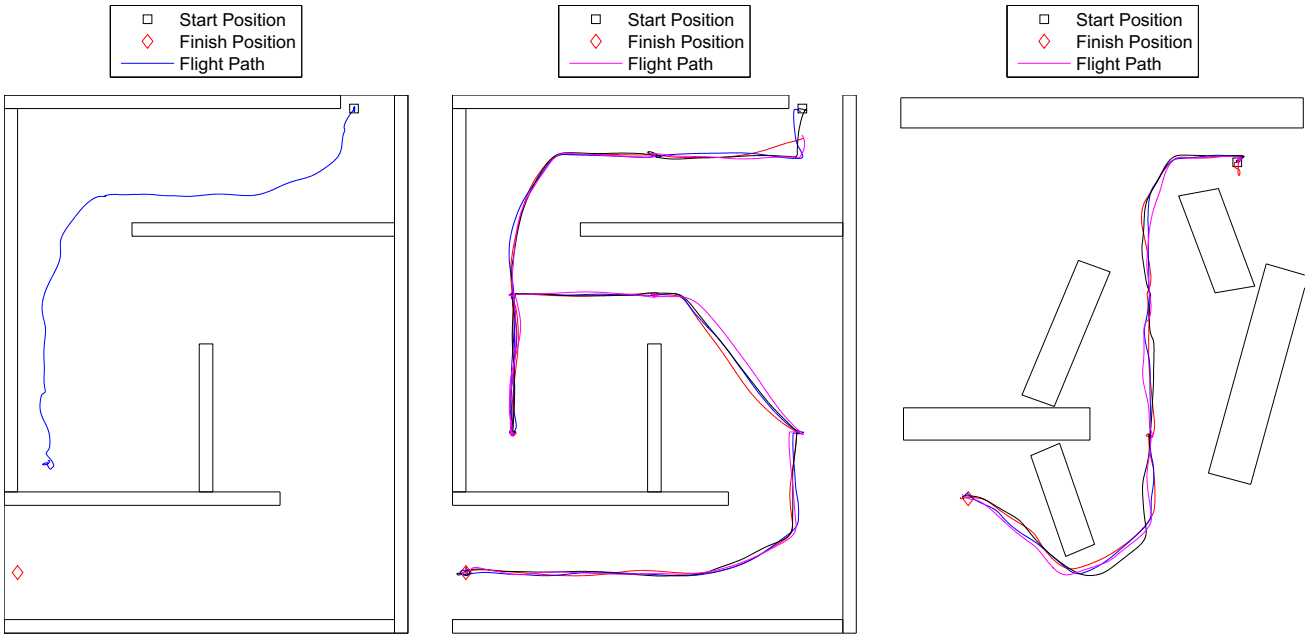
Fig. 7 Single axis Fast MPC simulation results

7.1.1 Simulation results for 1D

Initial results for the Fast MPC algorithm are presented in Fig. 7, where a simulated experiment was carried out on just the x-axis of the AR.Drone. The simulated plant used the dynamics identified in Sect. 2. The position target was alternated between +1 m and -1 m every 10 s. A soft constraint on the velocity of 0.5 ms^{-1} was imposed and all position constraints were relaxed.

Figure 7 demonstrates that the MPC controller performed well, driving the simulated vehicle to the setpoints whilst adhering to the velocity constraints. The time taken to execute a single control update, or rather the *turnaround time*, had a mean value of 7.2 ms. Note that this figure was achieved by exploiting sparsity in the matrix structure, such as the patterns apparent in (8), (9) and (10) due to the decoupling of the axes dynamics. The version of code generator used did not exploit sparsity natively, so these were implemented in custom multiplier utilities. Without exploiting sparsity, turnaround time increased to 13.6 ms, almost doubled, highlighting the importance of exploiting structure in the solver.

At the bottom of the Fig. 7 the residuals for the optimisation are plotted. It can be seen that every ten seconds the residuals spike, demonstrating that the optimisation has not converged to the optimal. This spiking occurs as the setpoints change and is due to the low iteration count used within Fast MPC. The initial solutions used are from the previous iteration due to the warm start procedure, which will be far from the optimal solution when the setpoint changes significantly. This can be seen to have minimal, if any, impact on



(a) Obstacle field 1 without learning **(b)** Obstacle field 1 **(c)** Obstacle field 2

Fig. 8 Paths taken by MAV in example scenarios

performance as the residual falls back down very quickly—usually after just one further iteration.

7.1.2 Decoupling the control axes

In order for the quadrotor to navigate environments such as the examples depicted in Fig. 1, it is necessary for the MPC to operate on both the x and y axes. Due to the symmetry in the vehicle's dynamics it is possible to decouple the x and y control into two separate MPC controllers. By decoupling the control axes it should be possible to reduce the computational complexity of the problem, although the representation of the operating region (for obstacle avoidance) must be simplified as demonstrated in Sect. 4.1.

First the single axis MPC controller used above was duplicated within the controller, such that the x and y axes were being optimized for separately. It was found that solutions did not change, as would be expected, but the turn around time did increase. The turnaround time increased from 7.2 ms for the single axis to 17.9 ms for the dual axis control. It is interesting that the turnaround time increased by a factor larger than two, as the number of operations required by the dual MPC algorithms is exactly double.

By solving for both axes within a single MPC optimizer, the turnaround time could be expected to take longer than double due to non-linear scaling of the matrix operations. The mean turnaround time was 43.9 ms in the combined case, showing that it is considerably more expensive. The computational cost may be worth it, however, as constraints may be posed in terms of both the x and y axes, allowing non-orthogonal operating regions to be represented. Crucially, this value indicates that real time operation at 10 Hz is feasible.

7.1.3 Initial flight test results

A flight test result is shown in Fig. 8a where the MPC controller was used to control the quadrotor, but the learning algorithm was disabled. In this example the setpoint presented to the MPC is the closest point within the operating region to the goal. It can be seen that the quadrotor hugs the walls as it is attracted to the goal and finally ends up getting stuck down a dead end. The walls are artificially enlarged when sent to the function that computes the operating region, taking into account the size of the vehicle as well as some buffer for violation of the position soft constraints. The resulting path is therefore not seen to directly touch the walls. Given the orthogonal nature of the obstacles, the first convexification method of Sect. 4.1 was used to determine the operating regions

7.2 Results using MPC and learning

The same scenario from the previous section was flown, but this time with the addition of the reinforcement learning exploration algorithm presented in Sect. 6. Figure 8 shows four separate flight paths taken for each of the presented scenarios, demonstrating good repeatability. In Fig. 8b the

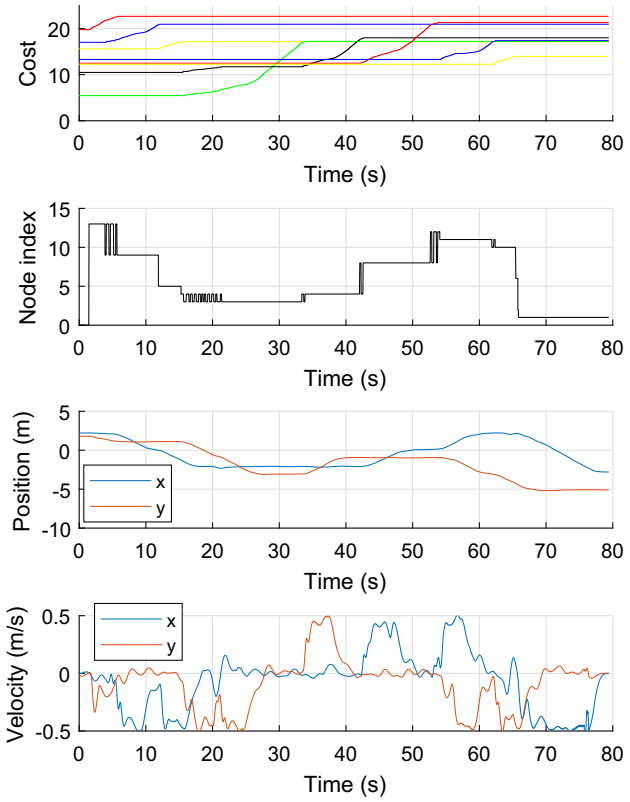


Fig. 9 Node costs, aim index, position and velocity histories for obstacle field 1

obstacles are all aligned with the axes and the rectangular convexification technique from Sect. 4.1 was employed, together with decoupled control for each of the two axes of motion. Figure 8c shows the flight path taken for the second scenario in which obstacles are not orthogonally aligned and the second method for computing operating regions from Sect. 4.2 was required. In this case the slower combined MPC for both axes was employed, but still running in real time. A video of the experiments seen in Fig. 8b can be found at https://youtu.be/Vym7QEdG7OM.

The effect of the reinforcement learning can particularly be seen in Fig. 8b where the MAV takes a seemingly direct route past the obstacles towards the goal, but gets stuck in a dead end, requiring it to turn around. After hovering for a few seconds, the reinforcement learning drives up the cost of the learning node and finds it to be more favourable to return to the previous node. After backtracking, the MAV then takes an alternate route around the walls in the centre of the obstacle field and makes its way to the finish position.

Figure 9 shows how the cost of the nodes varied over the duration of one of the flights, whilst learning to navigate the first obstacle field. For brevity the node indices and where they map to in physical space are not shown; rather the plot illustrates how the cost of nodes increase one at a time depending on which is being aimed for. The figure also shows the time history of the node index being aimed for and the resulting position traversed by the MAV on its way to the goal. One interesting observation here is the occasional rapid switching between two node indices, here the cost of the pair of nodes plus the estimated cost to reach them (Eq. 31) is very similar. During this period of switching rapidly between target nodes the costs of both nodes increase until a distinct new node is selected.

The velocity history in Fig. 9 shows that the MAV navigates almost continually through the environment, close to its upper velocity magnitude of 0.5 m/s. As discussed in Sect. 5, despite the constraint for each plan to terminate in a stationary condition, the MAV itself rarely stops. An exception to this can be seen around 30 s into the flight: at this point, the MAV has reached a node that continues to have the lowest cost, which is the aforementioned dead end of obstacle field 1. This dead end node with cost coloured green (Fig. 9) must increase until the node in black (backtracking) becomes the best next choice. The rate at which nodes increase in cost can be tuned, as previously mentioned, through the weighting α in Eq. 33.

8 Conclusions and further work

A new method for exploring unknown environments with a quadrotor was introduced. It was shown that reinforcement learning could be used to navigate non-convex obstacle fields without maintaining a global map of the world. Fast model predictive control was executed in real-time on modest hardware for control, providing the necessary obstacle avoidance. Aspects of the MPC construction that affected solve time and obstacle representation were highlighted. Automatic code generation was used for construction of the controllers, which reduced development time, although special attention to matrix structure was necessary to maintain the speed of the MPC algorithm. The immediate extension for this work is to integrate with real sensing, such as an RGBD camera or LIDAR, and real localization such as SLAM. A more fundamental challenge is to make the placement of the learning nodes automatic within the environment, attaching them to recognized features rather than fixed locations. By attaching nodes to recognised features with a minimum level of detail (*e.g.* not blank walls) confidence in localisation can be maintained, much like the ideas presented in Bose and Richards (2013).

A Terminal equality constraints in fast MPC

This appendix describes how to add a set of ℓ_F terminal equality constraints $E_F x(k+T) = e_F$ to the Fast MPC formulation of Wang and Boyd (2010). Adopting the same notation as in that paper, starting from their (6), the problem is converted to a nonlinear optimization in the form

$$\text{minimize } z^T H z + g^T z + \kappa\phi(z) \tag{34}$$

$$\text{subject to } Cz = b \tag{35}$$

where z is an amalgamation of all the decision variables $z = (\Delta u(k)^T, x(k)^T, \ldots \Delta u(k+T-1)^T x(T)^T)^T$ and $\kappa\phi(z)$ is a barrier function representing the inequalities. With the added inequalities, this means that the matrices C and b now have the structures

$$C = \begin{bmatrix} -B & I & \cdots & 0 & 0 & 0 \\ 0 & -A & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -A & -B & I \\ 0 & 0 & \cdots & 0 & 0 & E_F \end{bmatrix}$$

$$b = \begin{bmatrix} Ax(k) + \hat{w}(k) \\ \hat{w}(k) \\ \vdots \\ \hat{w}(k) \\ e_F \end{bmatrix}$$

Note that the extra ℓ_F equality constraints in the problem will require a corresponding extra ℓ_F elements in the dual variable ν. The modified Schur complement $Y = C\Phi^{-1}C^T$ used to solve for the step in ν is given by

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & 0 & 0 & 0 \\ Y_{21} & Y_{22} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & Y_{T-1,T-1} & Y_{T-1,T} & 0 \\ 0 & 0 & \cdots & Y_{T,T-1} & Y_{TT} & Y_{FC} \\ 0 & 0 & \cdots & 0 & Y_{FC}^T & Y_{FF} \end{bmatrix}$$

where $Y_{FC} = \tilde{Q}_T E_F^T$ and $Y_{FF} = E_F Y_{FC}$ and all other submatrices are calculated as in Wang and Boyd (2010). Note that this retains the same sparsity pattern as the original Y so the same solution approach is followed: the Cholesky factor of Y is

$$L = \begin{bmatrix} L_{11} & 0 & \cdots & 0 & 0 & 0 \\ L_{21} & L_{22} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & L_{T-1,T-1} & 0 & 0 \\ 0 & 0 & \cdots & L_{T,T-1} & L_{TT} & 0 \\ 0 & 0 & \cdots & 0 & L_{FC} & L_{FF} \end{bmatrix}$$

where L_{FC} is found by solving $L_{TT}L_{FC} = Y_{FC}$ and then L_{FF} is found by the Cholesky factorization $L_{FF}L_{FF}^T = Y_{FF} - L_{FC}L_{FC}^T$. Then the solution of the equation $Y\Delta\nu = \beta$ can then proceed as in Wang and Boyd (2010) making use of the Cholesky factorization L of Y. The modifications to include the equality constraints are simply and extra step on the end of each stage in the process of solving for $\Delta\nu$.

References

Aswani, A., Gonzalez, H., Sastry, S. S., & Tomlin, C. (2013). Provably safe and robust learning-based model predictive control. *Automatica*, *49*(5), 1216–1226.

Augugliaro, F., Schoellig, A., & D'Andrea, R. (2012). Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 1917–1922).

Bali, C., & Richards, A. (2017). Robot navigation using convex model predictive control and approximate operating region optimization. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 2171–2176).

Bellingham, J. S., Richards, A. G., & How, J. P. (2002). Receding horizon control of autonomous vehicles. In *Proceedings of the American control conference*.

Borrelli, F., Subramanian, D., Raghunathan, A. U., & Biegler, L. T. (2006). MILP and NLP techniques for centralized trajectory planning of multiple unmanned vehicles. In *Proceedings of the American control conference*.

Bose, L. N., & Richards, A. G. (2013). Mav belief space planning in 3d environments with visual bearing observations. In *International micro air vehicle conference and flight competition (IMAV2013)*.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*(1), 14–23.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., et al. (2005). *Principles of robot motion: Theory, algorithms, and implementations*. Cambridge: MIT Press.

Cowling, I., Yakimenko, O., Whidborne, J., & Cooke, A. (2010). Direct method based control system for an autonomous quadrotor. *Journal of Intelligent & Robotic Systems*, *60*, 285–316.

Deits, R., & Tedrake, R. (2015). Computing large convex regions of obstacle-free space through semidefinite programming. In H. Levent Akin Nancy, M. Amato Volkan Isler & A. Frank van der Stappen (Eds.), *Algorithmic foundations of robotics XI* (pp. 109–124). Berlin: Springer.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., & Pollefeys, M. (2012). Vision-based autonomous mapping and exploration using a quadrotor mav. In *2012 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 4557–4564). IEEE.

Garcia, I., & How, J. (2005). Trajectory optimization for satellite reconfiguration maneuvers with position and attitude constraints. In *Proceedings of the 2005 American control conference, 2005* (pp. 889–894). IEEE.

Hartley, E. N., Jerez, J. L., Suardi, A., Maciejowski, J. M., Kerrigan, E. C., & Constantinides, G. A. (2014). Predictive control using an fpga with application to aircraft control. *IEEE Transactions on Control Systems Technology*, *22*(3), 1006–1017.

Hehn, M., & D'Andrea, R. (2011). Quadrocopter trajectory generation and control. In *Proceedings of the IFAC world congress*.

Helwa, M. K., & Schoellig, A. P. (2016). On the construction of safe controllable regions for affine systems with applications to robotics. In *2016 IEEE 55th conference on decision and control (CDC)*, (pp. 3000–3005).

Hoffman, G., Rajnarayan, D. G., Waslander, S. L., Dostla, D., Jang, J. S., & Tomlin, C. J. (2004). The stanford testbed of autonomous rotorcraft for multi-agent control (starmac). In *Proceedings of the 23rd digital avionics systems conference*.

Hoffmann, G., Waslander, S., & Tomlin, C. (2008). Quadrotor helicopter trajectory tracking control. In *AIAA guidance, navigation and control conference and exhibit, Honolulu, Hawaii* (pp. 1–14). Citeseer.

How, J., Bethke, B., Frank, A., Dale, D., & Vian, J. (2008). Real-time indoor autonomous vehicle test environment. *Control Systems, IEEE*, *28*(2), 51–64.

Kerrigan, E., & Maciejowski, J. (2000). Soft constraints and exact penalty functions in model predictive control. In *Control 2000 conference, Cambridge*.

LaValle, S. M., & Kuffner, J. J. (1999). Randomized kinodynamic planning. In *Proceedings of international conference on robotics and automation*.

Limon, D., Alvarado, I., Alamo, T., & Camacho, E. (2008). MPC for tracking piecewise constant references for constrained linear systems. *Automatica*, *44*(9), 2382–2387.

Liu, C., & Chen, W.-H. (2013). Hierarchical path planning and flight control of small autonomous helicopters using mpc techniques. In *Intelligent vehicles symposium (IV), 2013 IEEE* (pp. 417–422). IEEE.

Lu, F., & Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, *4*(4), 333–349.

Maciejowski, J. M. (2002). *Predictive control with constraints*. Englewood Cliffs: Prentice Hall.

Maeder, U., Borrelli, F., & Morari, M. (2009). Linear offset-free model predictive control. *Automatica*, *45*(10), 2214–2222.

Mataric, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, *8*(3), 304–312.

Michael, N., Mellinger, D., Lindsey, Q., & Kumar, V. (2010). The grasp multiple micro-uav testbed. *Robotics & Automation Magazine, IEEE*, *17*(3), 56–65.

Milam, M. B., Mushambi, K., & Murray, R. M. (2000). A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proceedings of the IEEE conference on decision and control* (pp. 845–851).

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313.

Nieuwenhuisen, M., Droeschel, D., Beul, M., & Behnke, S. (2014). Obstacle detection and navigation planning for autonomous micro aerial vehicles. In *2014 international conference on unmanned aircraft systems (ICUAS)* (pp. 1040–1047). IEEE.

Reif, J. H. (1979). Complexity of the movers problem and generalizations. In *20th IEEE symposium on the foundations of computer science* (pp. 421–427).

Richards, A. (2015). Fast model predictive control with soft constraints. *European Journal of Control*, *25*, 51–59.

Richards, A., & Boyle, P. (2010). Combining planning and learning for autonomous vehicle navigation. In *AIAA guidance, navigation, and control conference*.

Richards, A. G., & How, J. P. (2002). Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of American control conference*.

Schouwenaars, T., How, J. P., & Feron, E. (2004). Receding horizon path planning with implicit safety guarantees. In *Proceedings of the American control conference*.

Sharma, S. (2011). Qcqp-tunneling: Ellipsoidal constrained agent navigation. In *IASTED international conference on robotics*.

Sharma, S., & Taylor, M. E. (2012). Autonomous waypoint generation strategy for on-line navigation in unknown environments. *environment*, 2:3D.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1). Cambridge: Cambridge University Press.

Tatjewski, P. (2014). Disturbance modeling and state estimation for offset-free predictive control with state-space process models. *International Journal of Applied Mathematics and Computer Science*, *24*(2), 313–323.

Vitus, M., Pradeep, V., Hoffmann, G., Waslander, S., & Tomlin, C. (2008). Tunnel-milp: Path planning with sequential convex polytopes. In *AIAA guidance, navigation, and control conference*.

Wang, Y., & Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, *18*(2), 267–278.

Williams, B., Cummins, M., Neira, J., Newman, P., Reid, I., & Tardós, J. (2009). A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, *57*(12), 1188–1197.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *1997 IEEE international symposium on computational intelligence in robotics and automation, 1997. CIRA'97., Proceedings* (pp. 146–151). IEEE.

Yang, J., Shi, Y., & Rong, H.-J. (2016). Random neural q-learning for obstacle avoidance of a mobile robot in unknown environments. *Advances in Mechanical Engineering*, *8*(7), 1687814016656591.

Zhang, T., Kahn, G., Levine, S., & Abbeel, P. (2016). Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 528–535). IEEE.

Colin Greatwood is a postdoctoral researcher at the University of Bristol. He gained a MEng in Aerospace Engineering from the University of Bristol in 2008. He stayed at Bristol to obtain his Ph.D. in 2013 on nonlinear trajectory optimization. Now working with small unmanned air vehicles Dr Greatwood is interested in developing embedded control algorithms and machine learning techniques.

Arthur G. Richards received the M.Eng. degree from Cambridge University in 2000 and the SM and Ph.D. degrees from MIT in 2002 and 2004, respectively. Since 2004, he has been with the Department of Aerospace Engineering, University of Bristol, Bristol, U.K., where he is currently a Reader. His research interests include trajectory optimization, model predictive control, robotics, and their combination to develop high-performance guidance for autonomous vehicles.